Module 3: Strings and Input/Output

Topics:
• Strings and their methods
• Printing to standard output
• Reading from standard input

Readings: ThinkP 8, 10
Practice: Self study exercises
Strings in Python: combining strings in interesting ways

```python
s = "Great"
t = "CS116"
u = s + t
v = s + "!!!!!! " + t
w = s * 3
x = 2 * t
y = 'single quote works too'
z = 'strings can contain quotes" too'
```
Overloading of *

The following are all valid contracts of *:

*: Int Int -> Int
*: Int Float -> Float
*: Float Int -> Float
*: Float Float -> Float
*: Int Str -> Str
*: Str Int -> Str

What contracts apply to +? To -?
Other string operations

• Contains substring: \texttt{s in t}
  
  – Returns \texttt{True} if the string \texttt{s} appears as a substring in the string \texttt{t}
    
    "astro" in "catastrophe" \Rightarrow \text{True}
    
    "car" in "catastrophe" \Rightarrow \text{False}
    
    "" in "catastrophe" \Rightarrow \text{True}

• String length: \texttt{len(s)}
  
  – Returns the number of characters in string \texttt{s}
    
    \texttt{len(""')} \Rightarrow 0,
    
    \texttt{len("Billy goats gruff!"')} \Rightarrow 18
Extracting substrings

• $s[i:j]$ returns the substring from string $s$, containing all the characters in positions $i$, $i+1$, $i+2$, ..., $j-1$
• $s[i:j:k]$ steps by $k$, instead of 1
• $s[k]$ returns a string containing the character at position $k$
• Like Racket, strings in Python start from position 0

Suppose $s = "abcde"$, what strings are returned?

• $s[2:]$, $s[:3]$, $s[:$
• $s[1:5:2]$, $s[2::3]$, $s[::-1]$
• $s[4]$, $s[-1]$
Strings are immutable

We cannot change the individual characters in a string `s`

```python
s = "abcde"
s[3] = "X"  # causes an error
```

but

```python
s = s[:3] + "X" + s[4:]
```

returns a new string "abcXe" and assigns it to `s`

Note that `Int`, `Float`, `Str`, and `Bool` values are also immutable.
Methods in Python

• `str` is the name of a class in Python
• By convention, `Str` (capital S) is used in contracts
• Like the `math` module, `str` contains many functions, in its case to process string values, but do not `import` it
• To use the functions in `str`:
  ```python
s = "hi"
s.upper() ⇒ "HI"
```
• Note that none of the string methods modify the string itself
• The methods create new strings, or answer questions about strings
Partial listing of string methods

```python
>>> dir("abc")

[ ..., 'capitalize', 'center', 'count',
  'endswith', 'find', 'format',
  'index', 'isalnum', 'isalpha',
  'isdigit', 'islower', 'isspace',
  'isupper', 'join', 'lower',
  'lstrip', 'partition', 'replace',
  'rfind', 'split', 'startswith',
  'strip', 'swapcase', 'translate',
  'upper', ...]
```
Using string methods

```python
s = 'abcde 1 2 3 ab
```

What do the following calls return?

```python
s.find('a')
s.find('a',1)
s.startswith('abc')
s.count('a')
s.replace(' ','
', '')
s.strip()
```
Getting more information about a `str` method

```python
>>> help ("".isalpha)
S.isalpha() -> bool

Return True if all characters in S are alphabetic and there is at least one character in S, False otherwise.
```
Exercise

Write a Python function that consumes a non-empty first name, middle name (which might be empty), and a non-empty last name, and constructs a userid consisting of first letter of the first name, first letter of the middle name, and the last name. The userid must be in lower case, and no longer than 8 characters, so truncate the last name if necessary. For example,

```
userid("Harry", "James", "Potter") ⇒ "hjpotter"
userid("Ronald", "Bilius", "Weasley") ⇒ "rbweasle"
```
Recursive String definition and function template

A Python String is either
• "", or
• $s + t$, where $s$ and $t$ are strings, and $\text{len}(s)=1$.

```python
def my_string_template(s):
    if s=="":
        # base case
    else:
        # ... $s[0]$ ... my_string_template($s[1:]$)
```

First of string

Rest of string
Recursion on Strings

Write a Python function `str_score` that consumes a string `s`, and returns the score for `s`, where

- alphabetical characters are worth 1 point,
- digits are worth their numerical value, and
- any other character is worth 0.

For example,

```
str_score("CS 116") ⇒ 10
```
Run the following program in the Definitions window. What do you see?

def middle(a,b,c):
    largest = max(a,b,c)
    smallest = min(a,b,c)
    mid = (a+b+c) - largest - \
          smallest
    return mid

middle(10,20,30)
middle(0,10,-10)
middle(-1,-3,-2)
Python output:
printing information to the screen

x = 20
print(x)
print(x+5)
y = "dog"
print(y)
z = 42.8
print(z)
print(x,y,z)
More on `print` function

- Has an effect
  - printing to "standard output" - the screen
- Does not return a value
  - technically, we say it returns `None`
Displaying values in Python programs

• Interactions window, for variable `x`:

  ```python
  x
  print(x)
  ```

• Result *usually* looks the same (except for strings), but they are different

• Difference is obvious in Definitions window

  ➔ Need to use `print` in our programs to see results as the program is running
New: Functions do not always need to return values

• We can write a function which only prints data
• If a function does not include a `return val` statement, then the returned value (and type) is `None`
• The purpose statement does not need to include "Returns None" as this will be included in the contract.
Design recipe changes:
If a function includes `print` statements

- Include a description of what is printed in the **Purpose** statement
- Add a new section: an **Effects** statement (immediately after the purpose) to briefly indicate a value is printed
- **Examples** should include a description of the actual values printed for that input

It may also include a `return` statement.
Example: Write a function that prints a string three times – once per line

def print_it_three_times(s):
    '''prints s on three lines, once per line
    Effects: Prints to the screen
    
    print_it_three_times: Str -> None
    
    Example: Calling print_it_three_times("a")
    prints a once on each of three lines.
    '''

    print(s)
    print(s)
    print(s)
Testing Screen Output

• Our `check` module contains functions for two different approaches for testing screen output:
  – `set_screen` – sets things up for the tester to check screen output
  – `set_print_exact` – will check exactly whether screen output matches the expected screen output

• Unless told otherwise on an assignment, you can choose the testing approach that you prefer.

• In both cases, the screen output testing function must be called before the function to check the return value (`within` or `expect`).
Testing Screen Output: `set_print_exact`

- Include a string for each line of expected screen output:
  ```
  check.set_print_exact("CS 116",
                       "CS 116", "CS 116")
  ```
- Followed by appropriate `check` function to test value returned by the function (even if it is `None`)
- When `expect/within` runs, in addition to comparing expected and actual return values, the strings in `set_print_exact` will be compared to the actual strings printed by the function, line-by-line.
Example: Screen Output Only

```python
import check
def print_it_three_times(s):
    print(s)
    print(s)
    print(s)

# Q6 Test 1: a short string - "CS 116"
check.set_print_exact("CS 116", "CS 116", "CS 116")
check.expect("p3T1",
              print_it_three_times("CS 116"),None)
```

There is no `return`, so function returns `None`. This value is passed to `check.expect` to verify.
Test Output

**p3T1: PASSED**

`None` was correctly returned by our function.

**p3T1 - print output: PASSED**

The strings passed to `set_print_exact` are exactly the strings printed by the function.
A failed test

```python
def print_it_three_times(s):  # Incorrect
    print(s)
    print(s)
    print(s)
-----
None was correctly returned by our function.
```

p3T1: PASSED

p3T1 - print output: FAILED; expected
CS 116
CS 116
CS 116
saw
CS 116
CS 116

The function did not print the expected strings, so an informative message is printed.
Testing Screen Output: `set_screen`

- Give a description of expected screen output:
  
  ```python
  check.set_screen(
      "CS 116 on three lines"
  )
  ```

- Followed by appropriate `check` function to test value returned by the function (even if it is `None`)

- Test will print screen output along with your description of what the screen output should be.

- You must then compare the two.

- No comparisons of the actual and expected string outputs are made by the `check` module.

- Useful if the output is very complicated and long.
import check

def print_it_three_times(s):
    print(s)
    print(s)
    print(s)

# Q6 Test 1: a short string – "CS 116"
check.set_screen("CS 116 on three lines")
check.expect("Q6T1",
            print_it_three_times("CS 116"),None)

There is no return, so function returns None. This value is passed to check.expect to verify.
Test Output

QT1 PASSED

None was correctly returned by our function.

-----

QT1 (expected screen output):
CS 116 on three lines

QT1 (actual screen output):
CS 116
CS 116
CS 116

You must examine your output to see if it matches what you expected.
Printing vs Returning

Complete the full design recipes for $f_1$ and $f_2$.

def $f_1(x)$:
    print($x+1$)

def $f_2(x)$:
    return $x+1$
Debugging your program with \texttt{print} statements

• If you have an error in your program, place \texttt{print} statements at points throughout your program to display values of variables

• \textbf{IMPORTANT}: Remember to remove the \texttt{print} statements before submitting your code.

  — Your program may fail our tests, even if it returns the correct function values!!!
A new Python feature

- Python functions can use information received in three different ways –
  - Two ways we saw in Racket:
    - Parameters
    - Global constants
  - A new way:
    - Entered via the keyboard
User Input to a Python Program

```python
user_input = input()
```

• Program stops
• Nothing happens until the user types at keyboard
• When user hits enter, a string containing all the characters before the return is returned by `input`
• The string value is used to initialize the variable `user_input`
• Program continues with new value of `user_input`
More on user input

• Alternate form (preferred):
  
  `user_input = input(prompt)`

e.g.
  
  `city = input("Enter hometown:" )`

• Prints the value of `prompt` before reading any characters

• Value returned by `input` is always a `Str`
User Input and the Design Recipe

• When a function includes an `input` call, this must be described in the **Purpose** statement:
  – Describe what happens with the value entered by the user
  – Use parameter names in your description, where relevant

• It should also be mentioned (without the same detail) in the **Effects** statement.
def repeat_str():
    '''reads in a string s, and a number n, and
    prints s n times on one line
    Effects:
    * Two values are read in and
    * One string is printed
    repeat_str: None -> None
    Examples: If the user enters "abc" and 4
    when repeat_str() is called,
    "abcabcabcabc" is printed
    If the user enters "" and 100
    when repeat_str is called, "" is printed
    '''
A Simple Program using `input`

```python
def repeat_str():
    s = input("Enter string: ")
    t = input("Enter int>=0: ")
    n = int(t)
    print(n*s)
```
Testing With User Input

• Set the user inputs needed for the test in order.
• Always use strings for the input values.
• Include one string for each call to `input` that happens in your test:

```python
check.set_input("CS116","3")
```
• Follow with appropriate `check` function for value returned by the function
• Test function will automatically use these values (in order) when a value is expected from `input`
• You will be warned if the argument to `set_input` contains too few or too many values
Example: Test with User Input

```python
import check

def add_two_inputs():
    '''add_two_inputs: None -> Int'''
    x = int(input("Enter 1st integer: "))
    y = int(input("Enter 2nd integer: "))
    return x+y

# Test 1: two positive numbers
check.set_input("2","7")
check.expect("AddT1", add_two_inputs(), 9)
```
Example

Write the Python function `n_times` that reads a natural number `n` from the user via the keyboard, and prints out `n` once per line on `n` lines. The function returns `None`. 
More on strings:
Formatting screen output

• We can print strings
print("my dog has fleas")
• We can print integers
fleacount = 12
print(fleacount)
• We can even combine them
print("my dog has", fleacount,
     "fleas")
print("my dog has " +
     str(fleacount) + " fleas")
Creating formatted strings

The `format` method and placeholder `{}`

• We can describe the string we want to build, indicating where values should be inserted by using placeholders indicated by `{#}` inside the string

• Then supply the values to insert

```python
fleacount = 3
fleastring = "My dog has {0} fleas".format(fleacount)
print(fleastring)
```
The string you are building contains {{#}} and is followed by `.format(a_0, a_1, \ldots, a_n)`

- Uses the embedded {{#}} to show where a value should be inserted in the new string
- The # indicates which of the `format` arguments (0 – n) should appear at that location of the string

```python
s="Did {0} repay {1} ${2} from {0}'s pay?"
print(s.format("Tom", "Li", 20))
```
Examples

"I like {1}{0} {2}% of the time".format(116,"CS",500/6)
⇒ "I like CS116 83.33333333333333% of the time"

"I have taken {2}{0} and {2}{1}".format(115,116,"CS")
⇒ "I have taken CS115 and CS116."

"Temp is {0}C (or {1}F)".format(-10.0,(-10)*9/5+32)
⇒ "Temp is -10.0C (or 14.0F)"
Possible errors in formatting

• Incorrect number of values to insert

```python
>>> print("{0} {1} {2}".format(42.0, 12))
IndexError: tuple index out of range
```
Printing on one line

• Recall that
  
  ```python
  print("this goes","on","one line")
  print("this on the next")
  print("and so on")
  ```

  goes on three separate lines

• However,
  
  ```python
  print("this goes","on","one line",end="",")
  print("and this on the same",end="")
  print(" and so on")
  ```

  all goes on one line (due to value of end argument)
Special Characters

• So, we know how to use `print` statements to put information on one line

• Can you use a single print statement to put information over multiple lines?
  – Yes, but we need a special character `\n`

  ```python
  print("one line\nanother\nand another ")
  ```
  – Despite taking 2 characters to type, it counts as one in string length

  ```python
  len("A\nB\nC\n") ⇒ 6
  ```
Goals of Module 03

• You should be comfortable the following in Python:
  – Strings and their methods
  – Printing to the screen
  – Reading from the keyboard